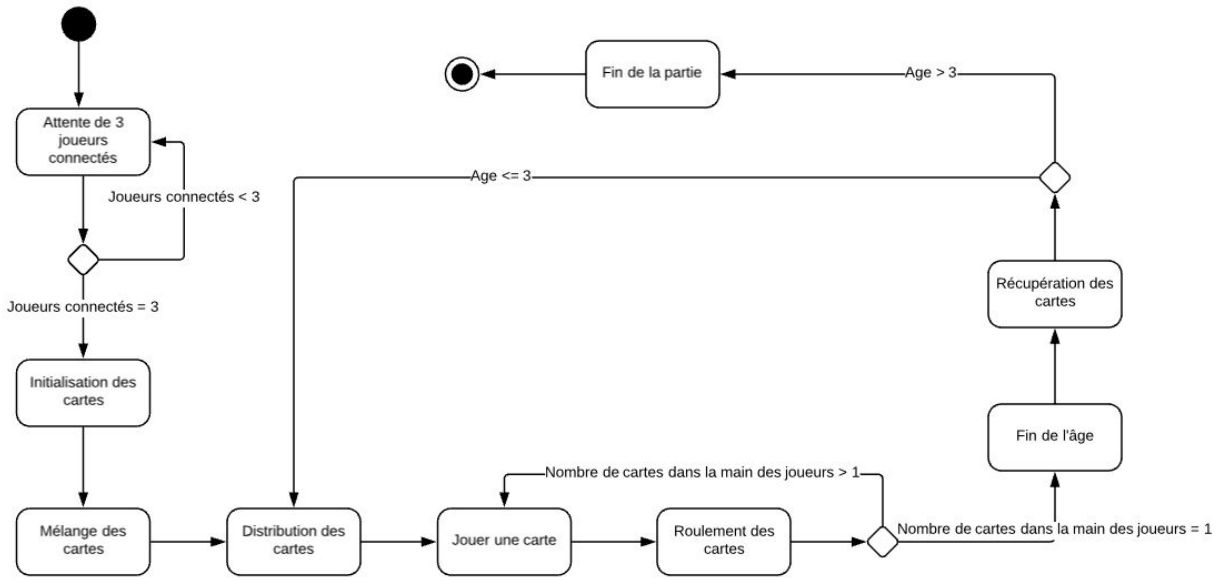
COO
Conception liée au Projet de Développement

1. Point de vue général de l'architecture

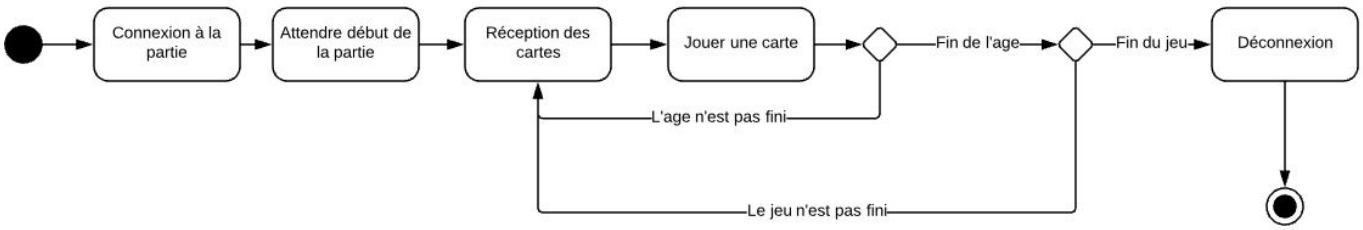
Représentation générale point de vue serveur :



Le serveur attend la connexion des joueurs, lorsque 3 joueurs se connectent la partie démarre.
Le serveur créer le jeu avec son initialisation et son mélange des cartes.
Puis il distribue les cartes aux joueurs.

Les joueurs jouent une carte puis échangent leurs cartes avec le joueur à gauche ou à droite en fonction de l'âge jusqu'au moment ou tous les joueurs ont seulement une seule carte en main.
Si il ne reste qu'une carte à chaque joueur alors c'est la fin de l'âge en cours et on récupère la carte non jouée.
Si l'âge n'est pas plus grand que 3 alors on refait les actions précédentes sinon c'est la fin de la partie.

Représentation générale point de vue client:



- Le client se connecte à une partie et attend que la partie se lance.
- Il réceptionne les cartes et joue une carte.
- Si l'âge n'est pas fini, il récupère les cartes lors du roulement et joue.
- Si l'âge est fini mais que la partie n'est pas terminée le joueur reçoit les cartes du nouvel âge.
- Si la partie est fini il se deconnecte.

2. Client

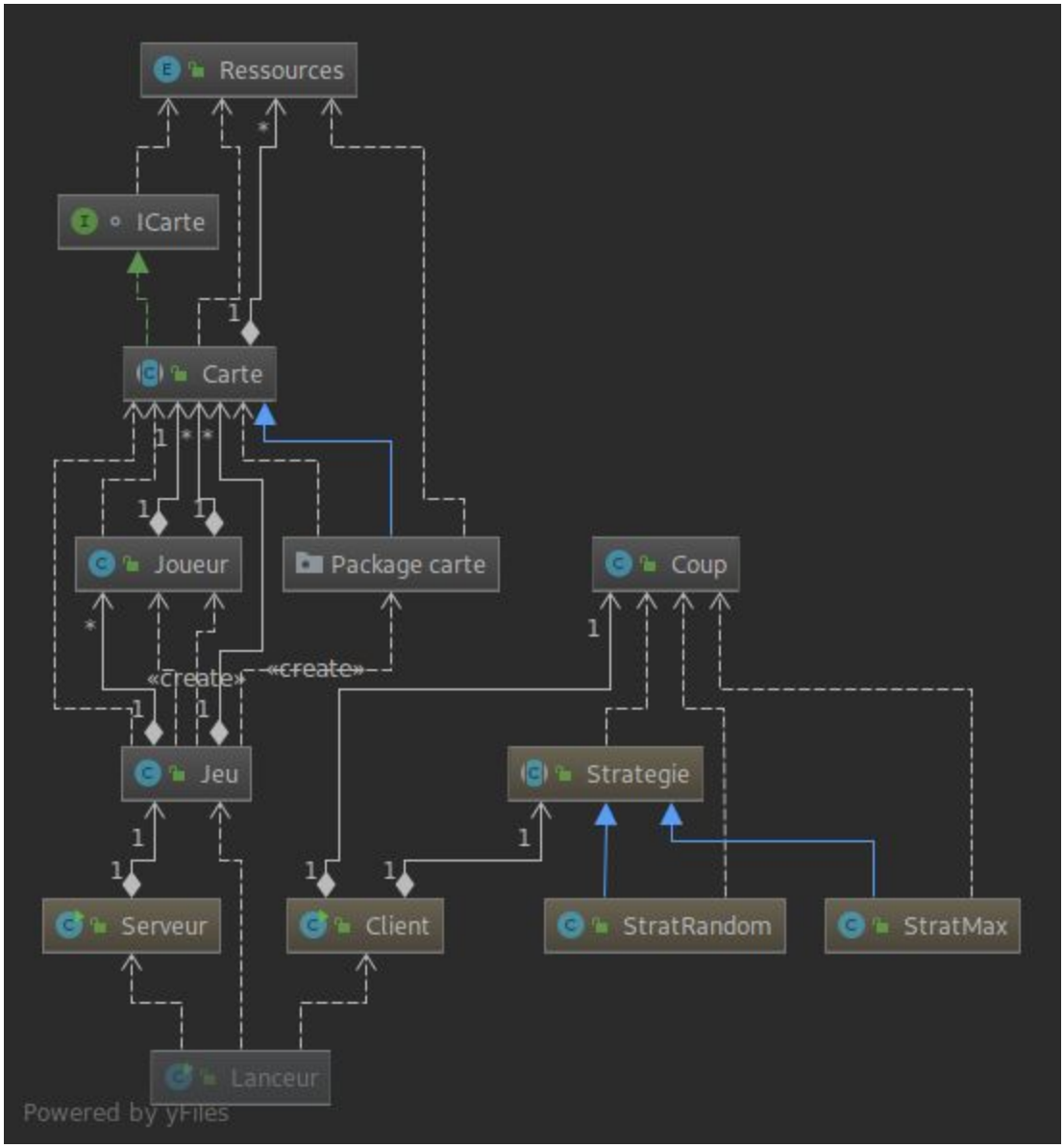
• *Analyse des besoins (exigences) : Cas d'utilisation*
    • Acteurs (le client est un acteur du serveur et vice versa)
    • Diagrammes de Cas d'utilisation
    • Scénarios (sous forme textuelle, un par use case)

• *Conception logicielle*

• Point de vue statique : Diagrammes de Classes lisible en format A4 (une découpe «logique » avec répétition de certaine classe).

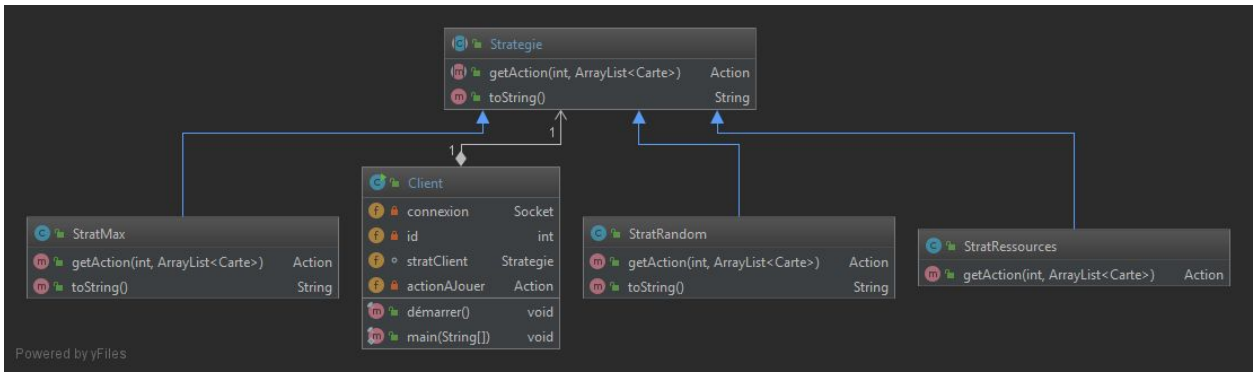
Point de vue Dynamique

Diagramme de classe général:



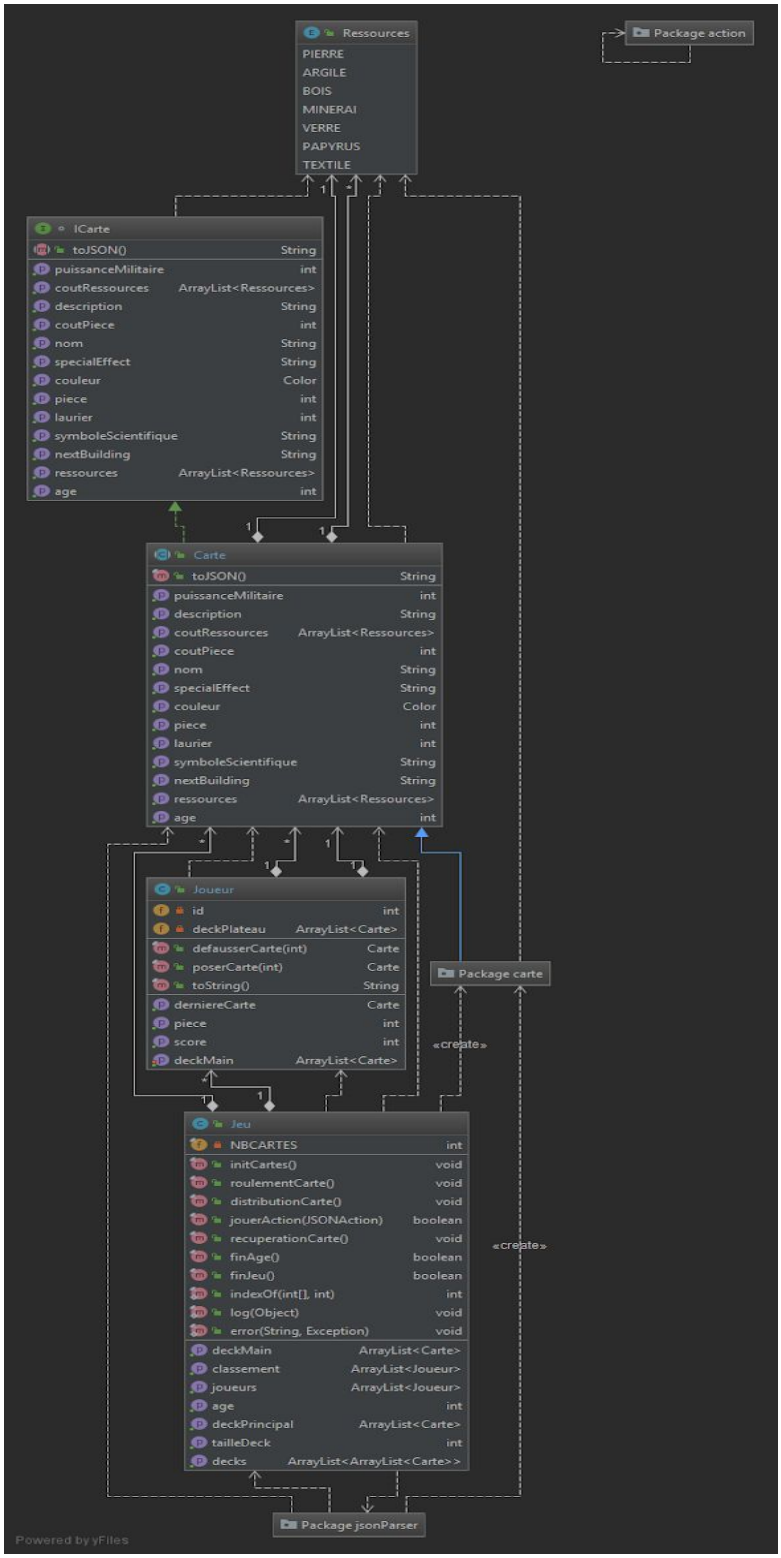
Un client ne peut avoir qu'une seule stratégie, et ne peut faire qu'un seul coup dans un tour. Le serveur n'est lié qu'à un seul jeu, et vice versa. Le jeu contient plusieurs joueurs, et plusieurs cartes. Celles-ci peuvent contenir 1 ou plusieurs bonus en ressources.

Diagramme du package client:



Un client (ou un bot, dans les faits) doit pouvoir se connecter via une socket. Chaque client possède un id spécifique qui permet leur identification par le serveur. Chacun a une action spécifique à faire à chaque tour, liée à une stratégie particulière.
La distinction en 3 stratégies permet théoriquement une action différente pour chaque client.

Diagramme du package moteur:



Dans le jeu, il y a différentes ressources: pierre, argile, bois...
Une carte peut posséder différentes caractéristiques: un coût en ressources ou en pièces à l'utilisation, une description et un nom unique. Lorsqu'on la joue, elle peut apporter un bonus variable en pièces, en puissance militaire, en ressources, ou en lauriers.

Selon le déroulement du jeu, une carte peut permettre la construction d'un certain "bâtiment", octroyant certains avantages selon des conditions.

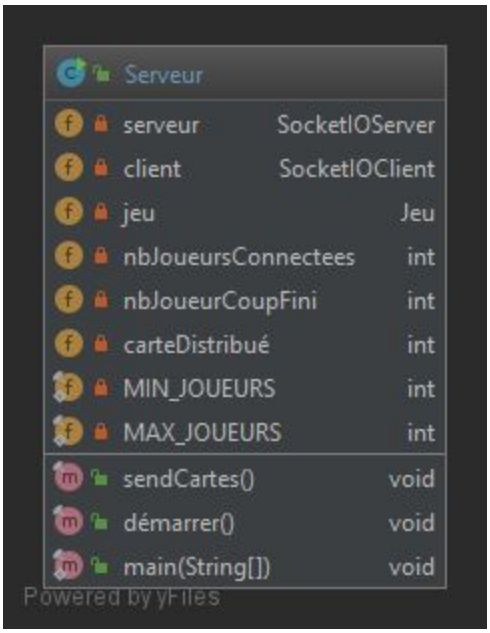
Enfin, plus la partie avance (avec le système d'âges), plus les cartes apportent des effets. Les classes abstraites ne sont pas itérables, contrairement aux interfaces, ce qui explique ce choix technique de notre part.

Un Joueur peut posséder une ou plusieurs cartes. Il a un id spécifique, permettant son identification, et possède son propre plateau. L'attribut "derniereCarte" nous permet de bien restituer les decks respectifs des joueurs (via une boucle).
Le joueur possède un certain nombre de pièces, un certain score, et un deck.
Durant un tour, il peut poser (jouer) une carte, ou en défausser une.

Un Jeu contient plusieurs joueurs. Il sait combien de cartes il faut distribuer.
D'une manière générale, il fait fonctionner le jeu, c'est à dire: initialiser les cartes, les distribuer, récupérer les cartes afin de faire tourner les decks des joueurs,
faire jouer un joueur. La méthode "indexOf", quant à elle, nous permet de retrouver une certaine carte dans le deck d'un joueur.
Le Jeu doit aussi avoir accès aux decks des joueurs, et aux joueurs eux-mêmes.

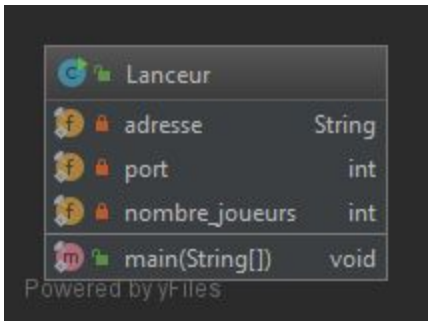
Enfin, nous utilisons JSON afin de faire circuler correctement les informations dans la liaison client-serveur.

Diagramme du package serveur:



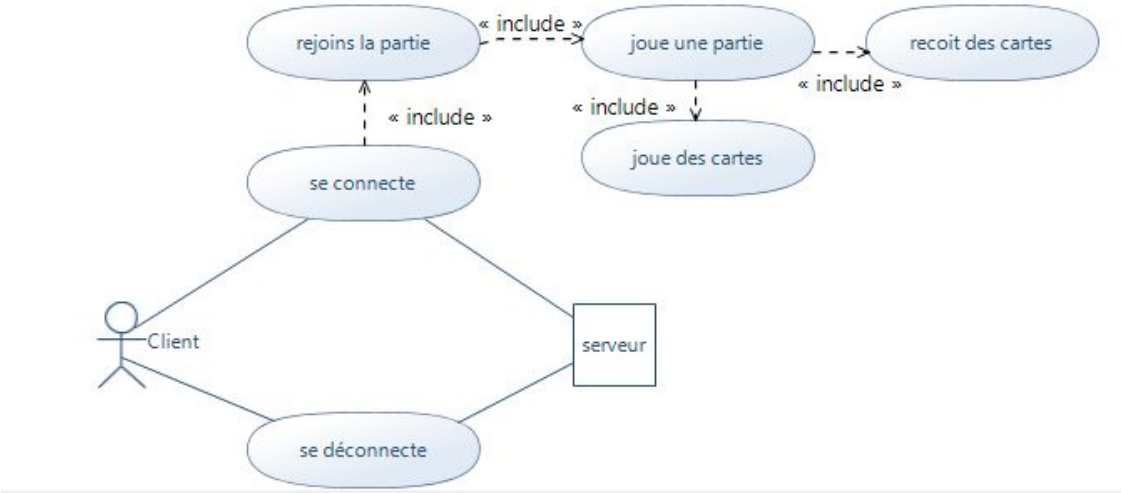
Notre serveur doit interagir avec les clients: contrôler le nombre de joueurs connectés afin de pouvoir commencer une partie, envoyer les decks aux joueurs, signaler le début et la fin des tours, des âges et du jeu avec toutes les opérations qui s'en suivent.

Diagramme du package lanceur:



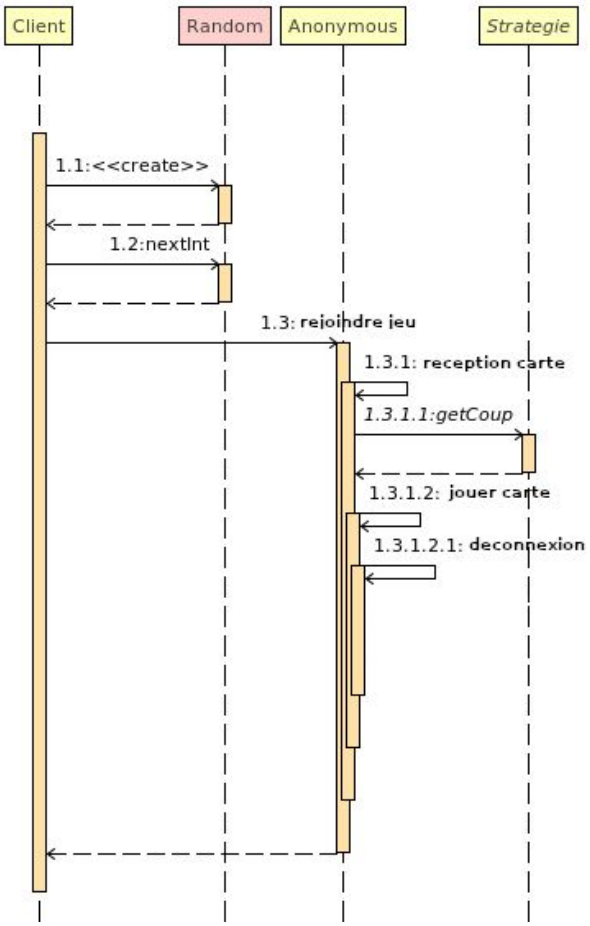
Le Lanceur doit lancer le serveur afin que les joueurs puissent s'y connecter. Il crée également nos joueurs (les bots), selon un nombre de joueurs qui lui est indiqué.
Pour faire tout cela, il doit posséder une certaine adresse pour lancer le serveur à cette adresse, et un port spécifique.

Use case :



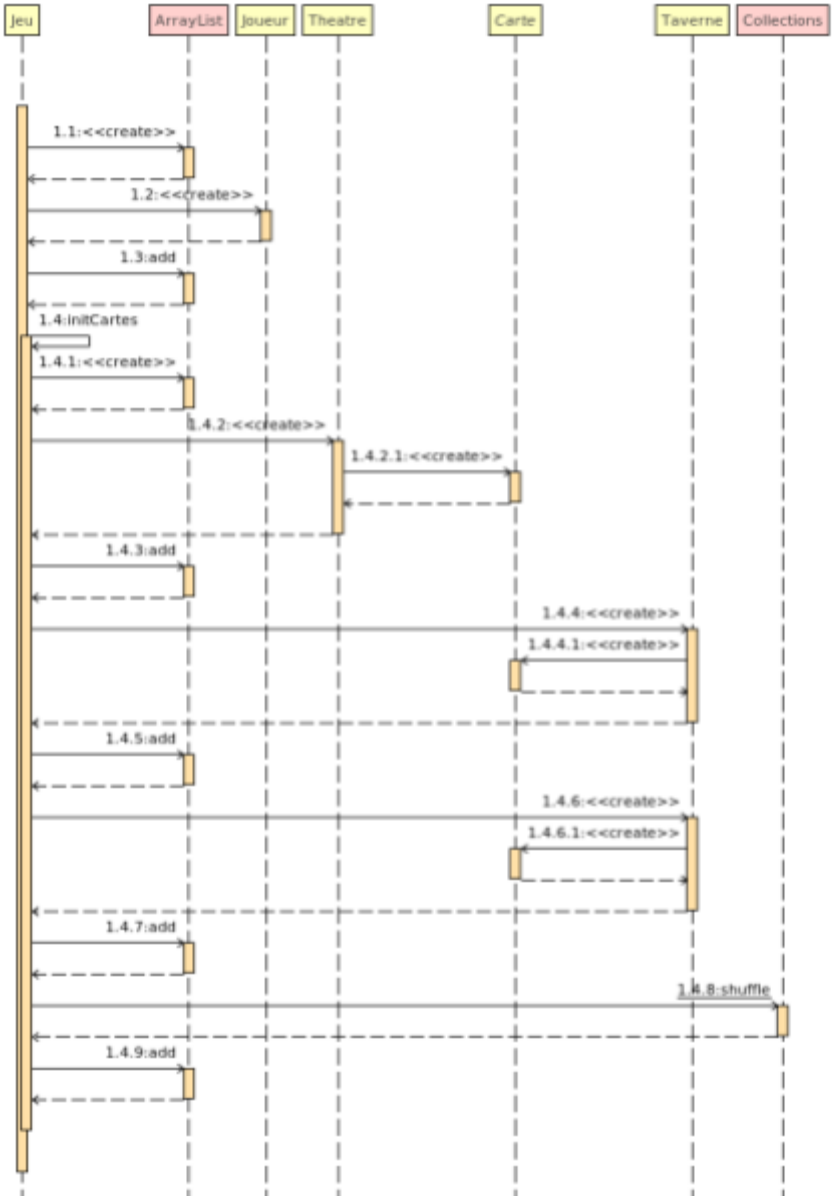
Le client peut se connecter et se déconnecté du serveur, la connexion lui permet de rejoindre de la partie (une fois le nombre de joueur requis atteint). Suite à cela, le client peut observer les bots jouer une partie ainsi que le déroulement de la partie (les bots reçoivent des cartes, jouent des cartes, …).

Diagramme de séquence:

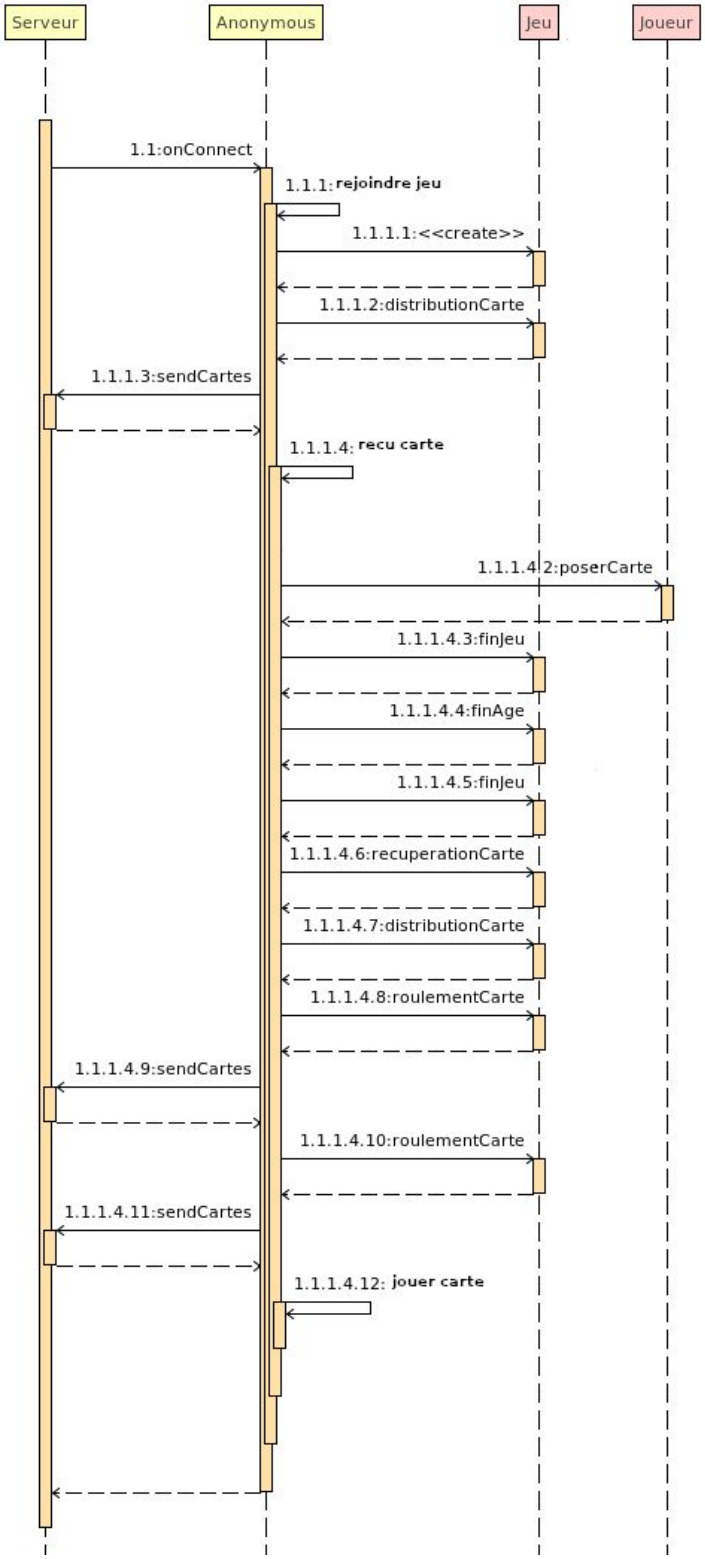


Avant de rejoindre le jeu, le client choisit une stratégie au hasard. Une fois avoir rejoint le jeu, le bot (anonymous) attend de recevoir les cartes pour demander à la classe Stratégie quelle carte jouer (getCoup) pour pouvoir ensuite la poser (jouer carte). Le bot peut se déconnecter (deconnexion) à tout moment.

On commence par créer une arrayList pour pouvoir ensuite créer un Joueur et l'ajouter à cette arrayList. On crée ensuite une nouvelle arrayList pour les cartes on crée suite une nouvelle carte (par exemple Théâtre qui va faire appel à la classe abstraite Carte) on l'ajoute à l'arrayList de carte. Une fois chaque carte ajoutée à l'arrayList elle est distribuée à la collection.



3. Serveur (même trame que pour le client)



Le serveur attend la connexion du bot (anonymous) une fois fait le bot créé une instance de la classe jeu et se fait distribuer les cartes. Il informe le serveur de son deck et le serveur attend que toutes les cartes soit distribué pour lancer de début du tour. Une fois que le bot a reçu l'information qu'il peut commencer à jouer (reçu carte) le bot commence à jouer et à chaque roulement de carte il informe à nouveau le serveur de sa main et attends l'autorisation du serveur pour commencer le prochain tour et ainsi de suite.

4. Conclusion

Points forts :

- Structuration du projet
- Client/Serveur sur un seul thread

Points faibles :

- Problème de communication avec le client

Évolution prévue :

- Ajout de toutes les cartes
- Intégration des autres règles
- Intégration et amélioration des IA